METHOD AND SYSTEM FOR IMPROVED PERFORMANCE OF A VIDEO GAME ENGINE

TECHNICAL FIELD

The present disclosure relates to the field of software tools for improving the performance of a video game engine.

BACKGROUND OF THE INVENTION

Most modern video game development is done using object oriented programming (OOP), wherein programming objects are used for each element of a game. The programming objects that represent elements within a game are often called game objects or game entities and are referred to herein as game objects. A game object can represent almost anything in a game, including characters, guns, treasures, trees, backgrounds, effects, etc. A game object is typically defined as an instance of an OOP class structure that includes methods and variables for the game object. Within computer memory, an OOP object (e.g., an instance of a class) is a structure that includes data and pointers to data in other locations within memory. For example, a game character might belong to a class that has values for position, orientation, size, mesh, etc., and also have methods defining behavior for the character. The memory location that contains the character game object includes data and can include pointers to other memory locations which contain more data for the character game object.

Current object oriented programing is not optimized for performance due in part to the use of reference value objects that contain pointers to data rather than containing data directly. Existing game development technology often uses reference value structures to define objects within a game. This is based on the concept of an object within the object oriented programming framework and is used for simplicity of programming (e.g., since the behavior and attributes of a programming object align well with those of a game object). However, object oriented programming may be optimized on a conceptual level and for ease of programming, but it is not always optimized for performance with respect to video game play. The main reason for the lack of optimized performance is that OOP programing does not automatically provide the optimum use of memory. OOP objects often contain pointers to data while the data itself is scattered randomly over distant memory locations. The result is that game object data is often in random places within memory and often contains pointers (e.g., to data) in other random locations within memory. In order to access the data for one or more characters (e.g., to determine the character location in a scene), a game engine will often have to access several separate random memory locations. There is also no hard guarantee of the relative location of data within memory for two different game objects. Accessing random memory locations for all game objects in a video game scene which runs at 60 frames per second (fps) or more is inefficient, especially considering the large amount of game objects which are typically in play during any given video game frame. Having game object data scattered over memory creates an inefficiency due to memory access time (e.g., the time it takes a central processing unit (CPU) to access a memory location, which is typically hundreds of CPU cycles each time a memory location is accessed). All memory accessing takes time; however, having to access memory in random distant locations requires additional time because the advantages of hardware prefetching are negated. The additional time it takes to access the scattered data within memory lowers the performance of executed game code at runtime. This puts limitations, for a given CPU speed, on the number of game objects that can be active in a frame during game play if a frame rate is to be maintained (e.g., 60 frames per second for typical games). This is particularly important for virtual reality applications which require 90 frames per second for minimum quality visual output. Modern game design improves performance by incorporating graphical processing units (GPUs) to offload processing from the CPU, as well as multithreaded coding techniques to parallelize the processing of game data over multiple CPU/GPU cores. However, these techniques do not overcome the fundamental issue of accessing separate random memory locations for game objects.

Game performance can also be improved by considering data oriented programming methodology as opposed to object oriented programming methodology, however, data oriented programming requires a high degree or knowledge for a game developer, and is done manually, and is specifically targeted to each game. This is out of reach for a large portion of game developers and game designers who have only a basic knowledge of programming methodology.

BRIEF DESCRIPTION OF THE DRAWINGS

Further features and advantages of the present invention will become apparent from the following detailed description, taken in combination with the appended drawings, in which:

FIG. **1** is a schematic illustrating an entity component system (ECS) device in an ECS system, in accordance with one embodiment;

FIG. **2**A is a schematic illustrating a memory layout for a chunk in an ECS system, in accordance with one embodiment;

FIG. **2**B is a schematic illustrating a memory layout within two chunks in an ECS system, in accordance with one embodiment;

FIG. **3** is a schematic illustrating a memory layout for a component data array within a chunk in an ECS system, in accordance with one embodiment;

FIG. **4**A is a schematic illustrating a method for integrating an entity into an archetype within an ECS system, in accordance with one embodiment;

FIG. **4**B is a schematic illustrating a method for creating an entity within an archetype within an ECS system, in accordance with one embodiment;

FIG. **4**C is a schematic illustrating a method for modifying an entity within an ECS system, in accordance with one embodiment;

FIG. **5** is a schematic illustrating a method for modifying entity data within an ECS system, in accordance with one embodiment;

FIGS. **6**A, **6**B and **6**C illustrate a method for deleting an entity within an ECS system, in accordance with one embodiment;

FIGS. **7**A and **7**B show a method for converting an object oriented programming object to an entity within an ECS system, in accordance with one embodiment;

FIG. **8** is a block diagram illustrating an example software architecture, which may be used is conjunction with various hardware architectures described herein; and

FIG. **9** is a block diagram illustrating components of a machine, according to some example embodiments, configured to read instructions from a machine-readable medium